TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

CHYASAL, LALITPUR

Lab Report No: -06

Title: - Virtual Functions

Submitted by: -

Name: - Diwas Pokhrel

Roll NO: - HCE081BEI014

Submitted To: -

Department of Electronics and Computer Engineering

Checked by: -

Date of submission: - 2082/03/21

OBJECTIVE

- To understand the concept of virtual functions in C++ and their role in achieving runtime polymorphism.
- To implement virtual functions in a class hierarchy to observe dynamic binding.
- To demonstrate the use of the virtual keyword in base and derived classes.
- To explore how virtual functions enable function overriding in inheritance.
- To analyze the behavior of virtual functions with different types of object pointers and references.
- To understand the significance of virtual destructors in preventing memory leaks.

THEORY

Virtual functions in C++ are a fundamental feature of object-oriented programming that facilitate runtime polymorphism, enabling a derived class to override a function defined in its base class. By declaring a function as virtual in the base class, the C++ runtime determines which function to call based on the actual type of the object, rather than the type of the pointer or reference used to access it. Virtual functions are critical in inheritance hierarchies, allowing derived classes to provide specialized implementations of a base class's interface. Additionally, virtual destructors are essential when deleting objects through a base class pointer to ensure proper cleanup of derived class resources, preventing memory leaks. Pure virtual functions, declared using = 0, make a class abstract, meaning it cannot be instantiated and must be inherited by a derived class that provides implementations for all pure virtual functions. Virtual functions introduce a slight performance overhead due to vtable lookups but provide significant flexibility and extensibility, enabling polymorphic behavior in complex programs.

Code Example

C++ program demonstrating virtual functions, virtual destructors, and a pure virtual function in a class hierarchy is given below :

```
#include <iostream>
using namespace std;
class Base {
public:
   // Virtual function
   virtual void display() {
      cout << "Display from Base class" << endl;
   }
```

```
                                                    // Pure virtual function
    virtual void pureVirtual() = 0;
                                                    // Virtual destructor
    virtual ~Base() {
        cout << "Base class destructor" << endl;
    }
};
class Derived : public Base {
public:
                                                    // Overriding the virtual function
        void display() override {
        cout << "Display from Derived class" << endl;
    }

                                                    // Implementing the pure virtual function
        void pureVirtual() override {
        cout << "Pure virtual function implemented in Derived class" << endl;
    }
    // Destructor
    ~Derived() {
        cout << "Derived class destructor" << endl;
    }
};
int main() {
    // Pointer of Base type pointing to Derived object
    Base* ptr = new Derived();

    // Call virtual function (resolves to Derived's display)
    ptr->display();

    // Call pure virtual function
    ptr->pureVirtual();
```

```
    // Delete object to invoke destructors
    delete ptr;

    return 0;
}
```

The output is :

```
Display from Derived class
Pure virtual function implemented in Derived class
Derived class destructor
Base class destructor
```

The display() function call through the base class pointer resolves to the Derived class's implementation due to the virtual keyword.

The pure virtual function pure Virtual () is implemented in the Derived class and called successfully.

The virtual destructor ensures that both the Derived and Base class destructors are called in the correct order, preventing resource leaks.